LAPORAN PRAKTIKUM 5
PEMROGRAMAN LANJUT
JAVA COLLECTIONS

Oleh

Nama	:	Rizqillah
NIM	:	1957301020
Kelas	:	TI 2C
Dosen Pembimbing	:	Musta'inul Abdi, SST., M.Kom.

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI DAN KOMPUTER

TAHUN 2021

LEMBAR PENGESAHAN

No. Praktikum	:	05/PPL/2C/TI/2021
Judul	:	Java Collections
Nama	:	Rizqillah
NIM / Kelas	:	1957301020 / TI 2C
Jurusan	:	Teknologi Informasi Dan Komputer
Prodi	:	Teknik Informatika
Tanggal praktikum	:	1 April 2021
Tanggal penyerahan	:	8 April 2021
Nilai	:	

Buketrata, 8 April 2021

Dosen Pembimbing,

Musta'inul Abdi, SST., M.Kom.
NIP. 19911030 20190310 1 5

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Tujuan

Modul ini mengenalkan suatu teknik pemrograman yang lebih tinggi. Dalam bagian ini Anda akan mempelajari Java Collections.

Setelah menyelesaikan pelajaran ini, diharapkan Anda dapat:

1. Memahami dan menggunakan Java Collections
2. Membuat dan mengelola Java Collections dengan baik
3. Memahami cara menggunakan Java Collections

1.2 Dasar Teori

Collections framework merupakan bentuk algoritma yang digunakan untuk merepresentasikan dan memanipulasi collections. Semua collections frameworks mengandung hal-hal berikut:

- **Interfaces:** memungkinkan collections dimanipulasi secara independen.
- **Implementations:** merupakan implementasi dari collection interfaces. Dan merupakan struktur data yang reusable.
- **Algorithms:** merupakan method-method yang dapat digunakan untuk melakukan proses komputasi tertentu, seperti searching (pencarian) dan sorting (pengurutan), terhadap objek yang meng-implement collection interfaces. Method-method pada Java Collections Framework adalah polymorphic: maksudnya **nama method** yang sama dapat digunakan pada collection interface yang sesuai. Algoritma pada Java Collections Framework memiliki fungsi yang reusable.

Java telah menyajikan Collection class dan interface yang lain, yang semuanya dapat ditemukan di java.util package. Contoh dari Collection classes termasuk LinkedList, ArrayList, HashSet dan TreeSet. Class tersebut benar-benar implementasi dari collection interfaces yang berbeda. Induk hirarki dari collection interfaces adalah collection interfaces itu sendiri. Sebuah collection hanya sebuah grup dari object yang diketahui sebagai elemennya sendiri. Collection memperbolehkan penggandaan/salinan dan tidak membutuhkan pemesanan elemen secara spesifik.

SDK(Software Development Kit) tidak menyediakan implementasi built-in yang lain dari interface ini tetapi mengarahkan subinterfaces, Set interfaces dan List

interfaces diperbolehkan. Sekarang, apa perbedaan dari kedua interface tersebut. Set merupakan collection yang tidak dipesan dan tidak ada penggandaan didalamnya. Sementara itu, list merupakan collection yang dipesan dari elemen-elemen dimana juga diperbolehkannya penggandaan. HashSet, LinkedHashSet dan TreeSet suatu implementasi class dari Set interfaces. ArrayList, LinkedList dan Vector suatu implementasi class dari List interfaces

1.3 Interface List

Interface **List** merupakan sub-interface dari interface Collection. Interface List digunakan untuk mengkoleksi data dalam **bentuk terurut** dan **memperbolehkan duplikasi**. Interface List menambahkan operasi yang berkaitan dengan posisi.

Dua class penting yang ada dalam Java Collections Framework yang meng-implement interface List adalah: **ArrayList** dan **LinkedList**. Jika perlu List yang dapat menambahkan dan menghapus data dimana saja (random access) maka dapat menggunakan ArrayList. Namun jika perlu List yang dapat menambah dan menghapus data di sekitar tengah-tengah dan mengaksesnya secara sekuensial, maka LinkedList adalah pilihannya.

ArrayList menyimpan data seperti array (diakses dengan index) namun ukurannya dapat bertambah secara fleksibel. Elemen yang dapat dimasukkan dalam ArrayList bisa bermacam-macam, termasuk null. ArrayList bisa disamakan dengan class Vector namun bedanya ArrayList ini unsynchronized. Operasi size, isEmpty, get, set, iterator, dan listIterator berjalan dalam waktu konstan dan lebih cepat dari LinkedList. ArrayList merupakan versi fleksibel dari array biasa. Yang mengimplementasikan List interface.

LinkedList merupakan implementasi dari algoritma LinkedList yang dipelajari dalam struktur data.

Adapun perbedaan LinkedList dan ArrayList sebagai berikut :

1. ArrayList secara internal menggunakan array dinamis untuk meyimpan elemen. Sementara LinkedList secara internal digunakan untuk menyimpan elemen dari list tertaut ganda.
2. Manipulasi dengan ArrayList cenderung lambat karena secara internal menggunakan sebuah array. Jika elemen lainnya dipindahkan atau dihilangkan dari Array, maka seluruh bit akan bergeser ke memori. Sementara pada LinkedList lebih cepat dibanding ArrayList karena menggunakan linked list ganda, sehingga tidak diperlukan sedikit pergeseran dalam memori.

3. Sebuah kelas ArrayList dapat menjadi seperti sebuah list karena hal tersebut hanya mengimplementasi List saja. Sementara kelas LinkedList dapat menjadi sebuah list dan antrian karena mengimplementasikan interface List dan Dequeue.
4. ArrayList lebih baik untuk menyimpan dan mengakses data, sementara LinkedList lebih baik untuk manipulasi data.

Adapun dalam keadaan apakah lebih baik menggunakan LinkedList atau ArrayList :

1. Operasi penyisipan dan penghapusan memberikan kinerja yang baik di LinkedList dibandingkan dengan ArrayList. Karenanya jika ada persyaratan untuk sering menambahkan dan menghapus data, maka LinkedList adalah pilihan terbaik.
2. Operasi pencarian (method get) cepat di Arraylist tetapi tidak di LinkedList jadi Jika ada lebih sedikit operasi tambah dan hapus dan lebih banyak persyaratan operasi pencarian, ArrayList akan menjadi pilihan terbaik Anda.

1.3.1 Linked List

LinkedList adalah bagian dari Java Collection yang ada dalam paket java.util. Kelas ini merupakan implementasi dari struktur data LinkedList yang merupakan struktur data linier dimana elemen tidak disimpan di lokasi yang berdekatan dan setiap elemen merupakan objek terpisah dengan bagian data dan bagian alamat. Elemen-elemen tersebut dihubungkan menggunakan pointer dan alamat. Setiap elemen dikenal sebagai node. Karena dinamik dan kemudahan penyisipan dan penghapusan, LinkedList lebih disukai daripada Array. Ini juga memiliki beberapa kelemahan seperti node tidak dapat diakses secara langsung sebagai gantinya perlu memulai dari head dan mengikuti link untuk mencapai node yang ingin diakses.

Poin penting tentang Java LinkedList adalah :

- Kelas Java LinkedList dapat berisi elemen duplikat.
- Kelas Java LinkedList mempertahankan urutan penyisipan.
- Kelas Java LinkedList tidak disinkronkan.
- Kelas Java LinkedList, manipulasi cepat karena tidak perlu terjadi pergeseran.
- Kelas Java LinkedList dapat digunakan sebagai list, tumpukan atau antrian.

Adapun Method LinkedList :

1. Menambahkan Elemen : Untuk menambahkan elemen ke LinkedList, bisa menggunakan method add(). Adapun contoh dari method add adalah sebagai berikut ini :

- **add (Object):** Method ini digunakan untuk menambahkan elemen di akhir LinkedList.
- **add (int index, Object):** Method ini digunakan untuk menambahkan elemen pada indeks tertentu di LinkedList.

2. Mengubah Elemen : Setelah menambahkan elemen, jika ingin mengubah elemen, dapat dilakukan dengan menggunakan method **set().** Karena LinkedList memiliki index, maka elemen yang ingin kita ubah direferensikan oleh index elemen. Oleh karena itu, method ini mengambil index dan elemen yang diperbarui yang perlu disisipkan pada index tersebut.
3. Menghapus Elemen: Untuk menghapus elemen dari LinkedList, dapat menggunakan method remove(). Method ini memiliki beberapa macam contoh pemanggilan dan parameter, yaitu sebagai berikut :
   - **remove(Objek):** Method ini digunakan untuk menghapus objek dari LinkedList. Jika ada beberapa objek seperti itu, kemunculan pertama objek tersebut akan dihapus.
   - **remove(int index):** Karena LinkedList memiliki index, maka method ini mengambil nilai integer yang hanya menghapus elemen yang ada di indeks tertentu di LinkedList. Setelah menghapus elemen, semua elemen dipindahkan ke kiri untuk mengisi ruang dan indeks objek diperbarui.
4. Iterasi LinkedList : Ada beberapa cara untuk melakukan iterasi melalui LinkedList. Cara yang paling terkenal adalah dengan menggunakan perulangan for dasar yang dikombinasikan dengan method **get()** untuk mendapatkan elemen pada indeks tertentu dan perulangan for lanjutan.
5. Membersihkan LinkedList : pada LinkedList, method yang bisa digunakan untuk menghapus semua elemen LinkedList adalah dengan method clear().

Dan adapun Method lainnya seperti :

1. addAll(int index, Collection<E> c) **:** Method ini Menyisipkan semua elemen dalam koleksi yang ditentukan ke dalam list, dimulai dari posisi yang ditentukan.
2. addFirst(E e) : Method ini Menyisipkan elemen yang ditentukan di awal list.
3. addLast(E e) : Method ini Menyisipkan elemen yang ditentukan di akhir list.
4. get(int index) : Method ini mengembalikan elemen pada posisi yang ditentukan dalam list.
5. getFirst() : ini mengembalikan elemen pada posisi pertama dalam list.
6. getLast() : ini mengembalikan elemen pada posisi terakhir dalam list.
7. peek() : Method ini akan mengembalikan nilai head dari list. Akan tetapi tidak akan menghapus nilai.

8. pop() : Method ini akan mengembalikan nilai menggunakan fungsi seperti stack, dan nilai tersebut akan dihapus.
9. push() : Method ini akan mengisi data pada LinkedList seperti operasi pada Stack.
10. size() : Method ini akan mengembalikan nilai dari jumlah data dalam list.

1.3.2 ArrayList

ArrayList adalah bagian dari Java Collection dan berada dalam paket java.util. ArrayList memberi array dinamis di Java Collection. Meskipun, ini mungkin lebih lambat dari array standar tetapi dapat membantu dalam program yang membutuhkan banyak manipulasi dalam array.

Kelas Java ArrayList menggunakan array dinamis untuk menyimpan elemen. Ini seperti sebuah array, tetapi tidak ada batasan ukuran. ArrayList dapat menambah atau menghapus elemen kapan saja. Jadi, ini jauh lebih fleksibel daripada array tradisional. Ini dapat ditemukan di paket java.util. ArrayList ini seperti Vektor di C++.

ArrayList di Java juga dapat memiliki elemen duplikat. Ini mengimplementasikan interface list sehingga dapat menggunakan semua method interface list. ArrayList mempertahankan urutan penyisipan secara internal.

Poin Penting dari ArrayList :

- ArrayList mewarisi kelas AbstractList dan mengimplementasikan interface List.
- ArrayList diinisialisasi oleh ukurannya. Namun, ukurannya bertambah secara otomatis jika koleksi bertambah atau menyusut jika objek dikeluarkan dari koleksi.
- Java ArrayList memungkinkan kita mengakses list secara acak.
- ArrayList tidak dapat digunakan untuk tipe primitif, seperti int, char, dll. Maka akan membutuhkan kelas pembungkus untuk kasus seperti itu.
- ArrayList di Java dapat dilihat sebagai vektor di C ++.
- ArrayList tidak disinkronkan. Kelas tersinkronisasi yang setara di Java adalah Vector.

Bagaimana ArrayList bekerja secara internal ?

Karena ArrayList adalah array dinamis dan tidak perlu menentukan ukurannya saat membuatnya, ukuran array otomatis bertambah saat kita menambahkan dan menghapus item secara dinamis. Meskipun implementasi library sebenarnya mungkin lebih kompleks, berikut ini adalah ide yang sangat mendasar yang

menjelaskan cara kerja array saat array menjadi penuh dan jika mencoba menambahkan item :

- Membuat memori berukuran lebih besar pada memori tumpukan(misalnya memori berukuran ganda).
- Menyalin elemen memori saat ini ke memori baru.
- Item baru yang ditambahkan sekarang karena ada memori yang lebih besar yang tersedia sekarang.
- Hapus memori lama.

Method ArrayList :

1. Menambahkan Elemen: Untuk menambahkan elemen ke ArrayList, bisa menggunakan method add(). Method ini beberapa jenis, yaitu sebagai berikut:
   - add (Object): Method ini digunakan untuk menambahkan elemen di akhir ArrayList.
   - add (int index, Object): Method ini digunakan untuk menambahkan elemen pada indeks tertentu di ArrayList.
2. Mengubah Elemen : Setelah menambahkan elemen, jika ingin mengubah elemen, dapat dilakukan dengan menggunakan method **set().** Karena ArrayList memiliki index, maka elemen yang ingin kita ubah direferensikan oleh index elemen. Oleh karena itu, method ini mengambil index dan elemen yang diperbarui yang perlu disisipkan pada index tersebut.
3. Menghapus Elemen: Untuk menghapus elemen dari ArrayList, dapat menggunakan method remove(). Method ini memiliki beberapa macam contoh pemanggilan dan parameter, yaitu sebagai berikut :
   - **remove(Objek):** Method ini digunakan untuk menghapus objek dari ArrayList. Jika ada beberapa objek seperti itu, kemunculan pertama objek tersebut akan dihapus.
   - **remove(int index):** Karena ArrayList memiliki index, maka method ini mengambil nilai integer yang hanya menghapus elemen yang ada di indeks tertentu di ArrayList. Setelah menghapus elemen, semua elemen dipindahkan ke kiri untuk mengisi ruang dan indeks objek diperbarui.
4. Iterasi ArrayList : Ada beberapa cara untuk melakukan iterasi melalui ArrayList. Cara yang paling terkenal adalah dengan menggunakan perulangan for dasar yang dikombinasikan dengan method **get()** untuk mendapatkan elemen pada indeks tertentu dan perulangan for lanjutan.
5. Membersihkan LinkedList : pada LinkedList, method yang bisa digunakan untuk menghapus semua elemen LinkedList adalah dengan method clear().
6. get(int index) : Method ini mengembalikan elemen pada posisi yang ditentukan dalam list.
7. size() : Method ini akan mengembalikan nilai dari jumlah data dalam list.

8. isEmpty() : Method ini akan mengecek apakah nilai dari ArrayList masih kosong atau tidak. Dan keluarannya adalah true atau false.
9. iterator() : Mengembalikan nilai iterator atas elemen dalam list dalam urutan yang benar.

1.4 Interface Set

Interface **Set** merupakan sub-interface dari interface Collection. Interface Set **tidak membolehkan duplikasi data** di dalam collection. Method yang ada dalam interface Set sama dengan interface Collection. Method paling penting pada interface Set adalah equals() yang digunakan untuk mengecek kesamaan objek.

Dua class penting yang ada dalam Java Collections Framework yang meng-implement interface Set adalah : **HashSet** dan **TreeSet**. **HashSet** merupakan class yang sering digunakan untuk menyimpan collection yang bebas duplikasi. Untuk efisiensi, objek yang ditambahkan dalam HashSet, perlu untuk menggunakan method hashCode(). **TreeSet** merupakan class yang sering digunakan untuk mengekstrak elemen dari collection dalam urutan tertentu. Agar TreeSet berjalan dengan baik, elemen yang ditambahkannya ke dalamnya harus dapat diurut. Terkadang lebih mudah untuk menambahkan data ke dalam HashSet baru kemudian dikonversi ke TreeSet agar mudah diurut.

Untuk mengoptimalkan ruang penyimpanan HashSet, maka kita dapat melakukan tuning initial capacity dan load factor. Class TreeSet tidak memiliki opsi tuning karena tree selalu dalam kondisi seimbang, dan memastikan performa log untuk proses insert, hapus dan query.

Perbedaan HashSet dan TreeSet :

1. **HashSet**
   - Kelas menawarkan kinerja waktu yang konstan untuk operasi dasar (menambah, menghapus, memuat dan ukuran).
   - Tidak menjamin bahwa urutan elemen akan tetap konstan seiring waktu
   - Kinerja iterasi tergantung pada *kapasitas awal* dan *faktor beban* HashSet.
     - Cukup aman untuk menerima faktor muatan default, tetapi Anda mungkin ingin menentukan kapasitas awal yang kira-kira dua kali ukuran yang Anda harapkan akan meningkat.

2. **TreeSet**
   - menjamin log (n) biaya waktu untuk operasi dasar (menambah, menghapus, dan memuat)

- menjamin bahwa elemen himpunan akan diurutkan (naik, alami, atau yang ditentukan melalui konstruktornya)
- tidak menawarkan parameter penyetelan untuk kinerja iterasi
- menawarkan beberapa method yang berguna untuk menangani set memerintahkan seperti first(), last(), headSet(), dan tailSet()lain-lain

3. **Poin-poin penting :**
   - Keduanya menjamin koleksi elemen yang bebas duplikat
   - Biasanya lebih cepat untuk menambahkan elemen ke HashSet dan kemudian mengonversi koleksi ke TreeSet untuk traversal yang diurutkan duplikat.
   - Tidak satu pun dari implementasi ini yang disinkronkan. Itu adalah jika beberapa utas mengakses satu set secara bersamaan, dan setidaknya satu utas mengubah set, itu harus disinkronkan secara eksternal.
   - LinkedHashSet dalam beberapa hal menengah antara HashSet dan TreeSet. Diimplementasikan sebagai tabel hash dengan daftar tertaut yang berjalan melewatinya, bagaimanapun, HashSet menyediakan iterasi penyisipan yang tidak sama dengan traversal yang diurutkan yang dijamin oleh TreeSet.

1.4.1 HashSet

Kelas HashSet mengimplementasikan interface Set, didukung oleh tabel hash yang sebenarnya merupakan instance HashMap. Tidak ada jaminan dibuat untuk urutan iterasi dari himpunan yang berarti bahwa kelas tidak menjamin urutan elemen yang konstan dari waktu ke waktu. Kelas ini mengizinkan elemen null. Kelas ini juga menawarkan kinerja waktu yang konstan untuk operasi dasar seperti tambah, hapus, isi, dan ukuran dengan asumsi fungsi hash menyebarkan elemen dengan benar di antara set.

Kerja internal dari HashSet : Semua kelas interface Set didukung secara internal oleh Map. HashSet menggunakan HashMap untuk menyimpan objeknya secara internal. Untuk memasukkan nilai di HashMap memerlukan pasangan nilai kunci, tetapi di HashSet, hanya memberikan satu nilai.

Penyimpanan di HashMap : Sebenarnya nilai yang dimasukkan ke dalam HashSet bertindak sebagai kunci untuk Objek peta dan untuk nilainya, java menggunakan variabel konstan. Jadi dalam key-value pair, semua nilainya akan sama.

Poin Penting dari HashSet :

- Menerapkan Set Interface.
- Struktur data yang mendasari HashSet adalah Hashtable.

- Saat menerapkan Set Interface, nilai duplikat tidak diperbolehkan.
- Objek yang Anda sisipkan di HashSet tidak dijamin akan disisipkan dalam urutan yang sama. Objek disisipkan berdasarkan kode hash mereka.
- Elemen NULL diperbolehkan di HashSet.
- HashSet juga mengimplementasikan interface Serializable dan Cloneable.

Method HashSet :

1. Menambahkan Elemen: Untuk menambahkan elemen ke HashSet, bisa menggunakan method add(). Namun, urutan penyisipan tidak disimpan di HashSet. Kita perlu mencatat bahwa elemen duplikat tidak diperbolehkan dan semua elemen duplikat diabaikan.
2. Menghapus Elemen : Nilai dapat dihapus dari HashSet menggunakan method remove().
3. Iterasi melalui HashSet : Iterasi melalui elemen HashSet menggunakan method iterator(). Yang paling terkenal adalah menggunakan for loop yang disempurnakan.
4. size() : Method ini akan mengembalikan nilai dari jumlah data dalam set.
5. isEmpty() : Method ini akan mengecek apakah nilai dari HashSet masih kosong atau tidak. Dan keluarannya adalah true atau false.
6. clear : digunakan untuk menghapus semua elemen HashSet.

1.5 TreeSet

TreeSet adalah salah satu implementasi terpenting dari interface Set di Java yang menggunakan Tree untuk penyimpanan. Urutan elemen dipertahankan oleh himpunan menggunakan pengurutan alami atau pembanding eksplisit disediakan atau tidak. TreeSet harus konsisten dengan equals() jika ingin mengimplementasikan interface Set dengan benar. Itu juga dapat dilakukan oleh pembanding yang disediakan pada waktu pembuatan TreeSet, tergantung pada konstruktor mana yang digunakan. TreeSet mengimplementasikan interface NavigableSet dengan mewarisi kelas AbstractSet.

Kelas Java TreeSet mengimplementasikan interface Set yang menggunakan tree untuk penyimpanan. Ini mewarisi kelas AbstractSet dan mengimplementasikan interface NavigableSet. Objek dari kelas TreeSet disimpan dalam urutan menaik.

Poin Penting TreeSet :

- Kelas Java TreeSet berisi elemen unik hanya seperti HashSet.
- Akses kelas Java TreeSet dan waktu pengambilan cukup cepat.
- Kelas Java TreeSet tidak mengizinkan elemen null.
- Kelas Java TreeSet tidak tersinkronisasi.

- Kelas Java TreeSet mempertahankan urutan menaik.

Method TreeSet :

1. Menambahkan Elemen : Untuk menambahkan elemen ke TreeSet, kita dapat menggunakan method add(). Namun, urutan penyisipan tidak dipertahankan di TreeSet. Secara internal, untuk setiap elemen, nilainya dibandingkan dan diurutkan dalam urutan menaik. Perlu dicatat bahwa elemen duplikat tidak diperbolehkan dan semua elemen duplikat diabaikan. Dan juga, nilai Null tidak diterima oleh TreeSet.
2. Mengakses Elemen : Setelah menambahkan elemen, jika ingin mengakses elemen, dapat menggunakan method bawaan seperti contains(), first(), last(), dll.
3. Menghapus Nilai : Nilai dapat dihapus dari TreeSet menggunakan method remove(). Ada berbagai method lain yang digunakan untuk menghapus nilai pertama atau nilai terakhir.
4. Iterasi melalui TreeSet : Ada berbagai cara untuk melakukan iterasi melalui TreeSet. Yang paling terkenal adalah menggunakan for loop yang disempurnakan.

Fitur TreeSet :

1. TreeSet mengimplementasikan interface Set. Jadi, nilai duplikat tidak diperbolehkan.
2. Objek di TreeSet disimpan dalam urutan yang diurutkan dan menaik.
3. TreeSet tidak mempertahankan urutan penyisipan elemen tetapi elemen diurutkan berdasarkan kunci.
4. Jika kita bergantung pada urutan pengurutan alami default, objek yang disisipkan ke tree harus homogen dan sebanding. TreeSet tidak mengizinkan untuk memasukkan objek heterogen. Ini akan memunculkan classCastException pada Runtime jika kita mencoba menambahkan objek heterogen.

Bagaimana TreeSet bekerja secara internal?

TreeSet pada dasarnya adalah implementasi dari pohon pencarian biner yang menyeimbangkan diri seperti Pohon Merah-Hitam. Oleh karena itu operasi seperti menambah, menghapus, dan mencari membutuhkan waktu O (log (N)). Alasannya adalah pada pohon self-balancing, dipastikan bahwa tinggi pohon selalu O (log (N)) untuk semua operasi. Oleh karena itu, ini dianggap sebagai salah satu struktur data yang paling efisien untuk menyimpan data terurut besar dan melakukan operasi padanya. Namun, operasi seperti mencetak N elemen dalam urutan yang diurutkan membutuhkan waktu O (N).

BAB II
PEMBAHASAN

2.1 LinkedList

Program :

```
import java.util.*;
public class LinkedListTest {
    public static void main(String[] args) {
        //Membuat Instance/Objek dari LinkedList
        LinkedList buah = new LinkedList();

        //Menambahkan Data pada Objek buah
        buah.add("Jeruk");
        buah.add("Jambu");
        buah.add("Apel");
        buah.add("Melon");
        buah.add("Semangka");
        buah.add("Nanas");
        buah.add("Sirsak");

        //Mencetak/Menampilkan Data
        System.out.println("Nama Buah : " + buah);

        //Menghitung Jumlah/Ukuran pada Objek LinkedList
        System.out.println("Jumlah Buah: " + buah.size());
    }
}
```

Hasil :

```
Nama Buah : [Jeruk, Jambu, Apel, Melon, Semangka, Nanas, Sirsak]
Jumlah Buah: 7
BUILD SUCCESSFUL (total time: 0 seconds)
```

Penjelasan :

Membuat objek dari class LinkedList bernama buah.

```
LinkedList buah = new LinkedList();
```

Melakukan penambahan data didalam linkedlist.

```
buah.add("Jeruk");
buah.add("Jambu");
buah.add("Apel");
buah.add("Melon");
buah.add("Semangka");
buah.add("Nanas");
buah.add("Sirsak");
```

Mencetak data yang ada didalam objek LinkedList.

```
System.out.println("Nama Buah : " + buah);
```

Mencetak ukuran panjang data dalam linkedlist.

```
System.out.println("Jumlah Buah: " + buah.size());
```

2.2 ArrayList

Program :

```
import java.util.ArrayList;
public class ArrayListTest {
        public static void main(String[] args) {
            ArrayList<String> names = new ArrayList<String>();
            names.add("Jim");
            names.add("Jack");
            names.add("Ajeet");
            names.add("Chaitanya");
            System.out.println(names);

            names.set(0, "Lucy");
            System.out.println(names);
    }
}
```

Hasil :

```
[Jim, Jack, Ajeet, Chaitanya]
[Lucy, Jack, Ajeet, Chaitanya]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Penjelasan :

Membuat objek dari class ArrayList dengan tipe data String dan bernama names.

```
ArrayList<String> names = new ArrayList<String>();
```

Menambahkan data kedalam ArrayList dengan method add.

```
names.add("Jim");
names.add("Jack");
names.add("Ajeet");
names.add("Chaitanya");
```

Menampilkan data dari objek names yang sekarang.

```
System.out.println(names);
```

Mengubah nilai pada index 0, dengan nilai baru “Lucy”. Dan mencetak nilai sekarang

```
names.set(0, "Lucy");
System.out.println(names);
```

2.3 HashSet

Program :

```
import java.util.*;
public class HashSetTest {
    public static void main(String[] args) {
        //Membuat Instance/Objek ArrayList Integer
        HashSet<Integer> data = new HashSet<Integer>();

        //Memasukan Nilai Default
        data.add(1);
        data.add(2);
        data.add(3);
        data.add(4);
        data.add(5);

        //Memasukan Nilai Duplukat/Yang Sama Dengan Nilai
Sebelumnya
        data.add(5);
        data.add(4);
        data.add(3);

        //Menampilkan Daftar Nilai
        System.out.println(data);
    }
}
```

Hasil :

```
[1, 2, 3, 4, 5]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Penjelasan :

Membuat Objek dari class HashSet, dengan nama data dan tipe data Integer.

```
HashSet<Integer> data = new HashSet<Integer>();
```

Memasukkan data kedalam HashSet.

```
data.add(1);
data.add(2);
data.add(3);
data.add(4);
data.add(5);
```

Mencoba memasukkan data yang sama.

```
data.add(5);
data.add(4);
data.add(3);
```

Menampilkan data dari HashSet.

```
System.out.println(data);
```

2.4 TreeSet

Program :

```
import java.util.TreeSet;
public class TreeSetExample {
    public static void main(String[] args) {
        TreeSet<String> ts = new TreeSet<String>();

        // Elements are added using add() method
        ts.add("A");
        ts.add("B");
        ts.add("C");

        // Duplicates will not get insert
        ts.add("C");

        // Elements get stored in default natural
        // Sorting Order(Ascending)
        System.out.println(ts);
    }
}
```

Hasil :

```
[A, B, C]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Penjelasan :

Membuat objek dari class TreeSet dengan tipe String dan bernama ts.

```
TreeSet<String> ts = new TreeSet<String>();
```

Menambah data kedalam TreeSet dengan method add.

```
ts.add("A");
ts.add("B");
ts.add("C");
```

Mencoba menambah data duplikat kedalam TreeSet.

```
ts.add("C");
```

Menampilkan data dalam objek ts.

```
System.out.println(ts);
```

Adapun alasan mengapa nilai C tidak tampil 2 kali adalah dikarenakan TreeSet akan mengabaikan nilai yang sama.

BAB III
PENUTUP

3.1 Kesimpulan

Interface List merupakan sub-interface dari interface Collection. Interface List digunakan untuk mengkoleksi data dalam bentuk terurut dan memperbolehkan duplikasi. Interface List menambahkan operasi yang berkaitan dengan posisi.

ArrayList menyimpan data seperti array (diakses dengan index) namun ukurannya dapat bertambah secara fleksibel. Elemen yang dapat dimasukkan dalam ArrayList bisa bermacam-macam, termasuk null.

LinkedList adalah bagian dari Java Collection yang ada dalam paket java.util. Kelas ini merupakan implementasi dari struktur data LinkedList yang merupakan struktur data linier dimana elemen tidak disimpan di lokasi yang berdekatan dan setiap elemen merupakan objek terpisah dengan bagian data dan bagian alamat. Elemen-elemen tersebut dihubungkan menggunakan pointer dan alamat.

Kelas HashSet mengimplementasikan interface Set, didukung oleh tabel hash yang sebenarnya merupakan instance HashMap. TreeSet adalah salah satu implementasi terpenting dari interface Set di Java yang menggunakan Tree untuk penyimpanan. Urutan elemen dipertahankan oleh himpunan menggunakan pengurutan alami atau pembanding eksplisit disediakan atau tidak.

Perbedaan antara List dan Set adalah Interface List dapat berisi elemen duplikat sedangkan Interface Set hanya berisi elemen unik.

DAFTAR PUSTAKA

[1] Thamura, Frans. *Modul JENI-intro3: Bab03 Teknik Pemrograman Lanjut*. JENI.

[2] Viska Mutiawani dan Kurnia Saputra. 11 Maret 2014. http://informatika.unsyiah.ac.id/. Informatika Unsyiah.

[3] Oracle. "Class LinkedList". https://docs.oracle.com/

[4] GeeksForGeeks. "LinkedList in Java". 05 Maret 2021. https://www.geeksforgeeks.org/

[5] javaTpoint. "Java LinkedList Class". https://www.javatpoint.com/

[6] Appkey. "Perbedaan ArrayList dan LinkedList". https://appkey.id/. 28 Januari 2021.

[7] GeeksForGeeks. "ArrayList in Java". 26 Juni 2020. https://www.geeksforgeeks.org/

[8] javaTpoint. "Java ArrayList Class". https://www.javatpoint.com/java-arraylist

[9] QAStack. "HashSet vs TreeSet". https://qastack.id/programming/

[10] javaTpoint. "Java HashSet Class". https://www.javatpoint.com/java-hashset

[11] GeeksForGeeks. "HashSet in Java". 09 September 2020. https://www.geeksforgeeks.org/hashset-in-java/

[12] javaTpoint. "Java TreeSet Class". https://www.javatpoint.com/java-treeset

[13] GeeksForGeeks. "TreeSet in Java with Example". 26 Juni 2020. https://www.geeksforgeeks.org/treeset-in-java-with-examples/

[14] Oracle. "Class TreeSet". https://docs.oracle.com/.

[15] Oracle. "Class ArrayList". https://docs.oracle.com/

[16] Oracle. "Class HashSet". https://docs.oracle.com/